ISOLEV: A Level Surface Cutting Plane Program for CFD Data

G. David Kerlick[1]

Report RNR-89-006, June 1989

Computer Sciences Corporation
NASA Ames Research Center
Moffett Field CA 94035, USA

Abstract. This paper describes a computer program called ISOLEV which allows interactive visualization of CFD scalar and vector functions by means of a user-specified sections of the data. The application is based on a "marching cubes" style table lookup for isosurfaces on hexahedral grid cells. The program supports Gouraud-shaded color maps of the data, surface-on-surface maps, and deformation surfaces of vector fields. The execution of the code is improved by presorting the cells in the database and maintaining an active set of cells to be rendered. The program is implemented in the C language under UNIX on Silicon Graphics 3000 and 4D series workstations, and makes use of Tristram's Panel Library.

[1]This work was supported by NASA Contract NAS 2-12961 to Computer Sciences Corporation for the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center.

June 30, 1989

Copies of this report are available from:
NAS Applied Research Office
Mail Stop T045-1
NASA Ames Research Center
Moffett Field CA 94035
(415)-694-5197

1. Introduction

Recently, there has been increasing interest in the visualization of three dimensional data sets [VISC87, FREN89]. This problem of *volumetric visualization* has two main approaches. The older approach uses graphical vectors and polygons to visualization scalar or vector fields in space. A newer method of "direct volume rendering" instead uses "voxels" (three-dimensional analogs of pixels), applies analogs of image processing transformations to them, and then renders the voxels directly to a frame buffer with no intermediate geometric stage.

Direct volume visualization is a 3-D analog to 2-D image processing and has more in common which image processing techniques than with traditional graphical techniques using vectors and polygons. There are two main approaches [UPSO88, WEST89] The first approach is a generalization of ray-casting which maps the image plane (screen) back on to the data (backward mapping, e.g. [LEVO88]). The other method (forward mapping) processes the the voxels and accumulates them into a frame buffer (e.g. [DREB88]).

In current direct volume rendering methods, nonuniform curvilinear grids and perspective transformations are handled with difficulty or not at all. While these limitations are not severe for many types of data (e.g. seismic or medical imagery) they are significant for computational fluid dynamics (CFD). CFD datasets are typically based on body-conforming curvilinear coordinates and encompass several decades of length scales. The extension of direct volume rendering techniques to CFD datasets is still an open problem.

The present paper is concerned with the first approach, using vectors and polygons, and investigates the extent to which some of the valuable features of direct volume rendering can be approximated by more traditional techniques. At present, vectors and polygons can be rendered in hardware at greater speed thanvoxels. Traditional volume rendering algorithms at present are too slow for interactive rendition. Moreover, the methods described here work naturally on curvilinear grids with varying length scales. Finally, the geometry of vectors and polygons make it easier to obtain *quantitative* information about the dataset, for example the area of an isosurface, or the volume contained between two of them. This information is difficult to obtain directly from a voxel representation.

Most of the useful aspects of volume rendering can be simulated by polygonal "surfaces on surfaces" in which one scalar function forms the surface (typically a cutting plane) and other scalar functions are represented by color or transparency. In this manner, some of the functionality of tools like the PIXAR Computer "cube tool" can be simulated and extended by means of polygon-rendering hardware augmented by efficient sorting and sweeping algorithms.

The ideas expressed here can also be used in time-dependent problems by using the methods at each timestep in an animation. Moreover, the idea of sweeping lower-dimensional manifolds through an N-dimensional database is a straightforward extensionof these ideas.

A software prototype called ISOLEV implements these ideas, and some color plates of screen images produced by this program are appended. This program is written in C and runs under UNIX on Silicon Graphics workstations. COpies of the software can be obtained by writing to the NAS Applied Research Office at the address on the cover of this report.

2. Approach

We consider the visualization of multi-dimensional datasets using graphical vectors and polygons. In particular we consider systems based on discretizations of 3-D continua such as finite-element and finite-difference CFD. These methods can also be used on voxel data which has been sampled at grid points, for example by measuring the photometric density.

2.1 Grid Planes

The C-plane program [LEVI88] and its descendant D-plane [TRIS88] are special purpose programs which act on CFD datasets in the PLOT3D format [PLOT89]. There the data set is the image in physical space of a regular lattice in computational space. These program sweep through the grid indices in a specified order, and visualize the data one computational plane at a time. They do no interpolation of the data beyond the implicit bilinear interpolation used by Gouraud shading hardware. This program, in constrast, is based fundamentally on interpolation of the data.

2.2 Isoscalar surfaces

An isoscalar surface, or *isosurface* for short, is a surface in space defined by a scalar function F of space. That is, one constructs the surface

$$F(\mathbf{r}) = \beta.$$

The function F is a *scalar function* of the spatial position **r** which defines the surface and the value β is the *isovalue* or *threshhold*). The function F is operated upon by a *geometric transfer function* which represents the numerical values of the function F as graphical constructs in space. Other transfer functions such as color or opacity may act on different scalar functions.

2.3 Surface-on-Surface

A "surface on surface" contour plot is one where a surface defined by a geometric transfer function, say F(**r**), is colored or textured with information defined another function, say G(**r**). [This is done non-interactively in the routine con4x4 in the ARCGraph Mesh Library [ARCG88], for example. These plots are hard to interpret without additional depth cueing or lighting models.

For example, one may constract the sonic surface (Mach Number = 1) and color it by static pressure to obtain a visualization of the pressures on a shock surface. However, since there is both spatial and shading information in the plot, the visualization is difficult to interpret, since color informationis used to convey information about the scalar field, and about the geometry of the isoscalar surface, by means of lighting and shading. One can use only fully saturated hues for the scalar and reserve saturation and value for the lighting model, but this limits the types of color maps that can be used.

2.4 Cutting Plane

If the function which defines the isosurface is a simple one, then we have more hope of disentangling geometric information from scalar field information. This is precisely the case with the "cutting-plane" algorithm developed here. In this case, one uses a simple linear function

$$F(\mathbf{r}) = (\mathbf{r}-\mathbf{r}_0) \cdot \mathbf{n} = \beta$$

where $\mathbf{n}$ is a normal to a plane and $\mathbf{r}_0$ is a reference point. This "cutting plane" provides much of the same information which is desired from volume visualization, but economizes on the number of graphical primitives which need to be rendered. A color transfer function is used on a second scalar variable, and results in what looks like a section of a continuously colored volume. A third transfer function can be added for opacity. Other simple functions, such as computational grid surfaces, spheres or cylinders, or distance from an aerodynamic surface, can also be used as geometric transfer functions.

3. Implementation

The method of this paper comprises three basic concepts. First, it is assumed that one has at hand a discretization of space into cells of similar topology. Second, a set of "bitmaps" or lookup tables corresponding to these cell shapes and the sign of the value of the vertex is constructed. The prototype for this is the "marching cubes" algorithm [WYVI86, LORE87]. Finally, one must have a way of arranging the individual cells so that they may be rendered efficiently into vectors and polygons. The output of this process is a set of triangles which can then be lit and shaded. Extensive use has been made of David Tristram's Panel Library [TRIS89], a public domain package of software buttons, sliders, and other types of actuators which can be incorporated into a program to give it interactive control. For example, the viewport parameters, the isoscalar levels, and the level surface orientation are all controlled by sliders.

3.1 Discretization

The discretization of space generally falls into two types. By far the most common is discretization into topological cubes, and it is this discretization which we use here. A topological cube is a figure containing 8 vertices, 12 linear edges, and 6 (generally nonplanar) faces arranged in a cube [Fig. 1] This topological class includes the exact cubes typical of volume renderings as well as rectilinear parallelpipeds and the more general computational hexahedra obtained from curvilinear coordinates such as those in the PLOT3D program [PLOT89]. For curvilinear grids composed of these cells one can order the cells by three coordinate indices I,J,K which represent the position of the cells in computational space.

In the present implementation, a separate set of surfaces are constructed in a representation of computational space (IJK-space), since this provides an overall look at the solution that can be of use to a numerical analyst. In particular, one can have an overview of the scalar fields at all length scales at once.

The second type of discretization to which these ideas apply is the tesselation of space into tetrahedral cells with data points at their vertices. The most common method of tesselation is the generalization of Delaunay triangulation to three dimensions. The tetrahedral elements of this tesselation have the property that the circumspheres of the cells contain no other vertices. Bookkeeping is somewhat more complicated for such triangulations, because of their more complicated connectivity. However, since the sweeps to be implemented require a resorting of cells, this is not a great disadvantage. Triangulation methods can be used with completely unstructured datasets.

3.2 Marching Cubes Bitmaps

Having discretized the flowfield into cells, we now operate on each cell independently, using tables patterned after [LORE87] which assign a unique isosurface topology to each cell depending upon whether the scalar value being contoured is greater than or less than the value at the cell vertices. For the 8-vertex hexahedral cell typical of PLOT3D, a representation of this topology can be preloaded into an 8-bit table with 256 entries. The corresponding 4-bit table for tetrahedra has 16 entries.

The present program is based on a program written by Martin Fouts [personal communication] which implemented the marching cubes algorithm on a rectangular cube. Since the information contained in the table is topological, the same set of maps will do for the more general hexahedron. Fouts's original code stored two separate datasets at each bitmap, a pattern and a permutation of indices that rotates the pattern to the appropriate location. In the present implementation, all 256 patterns are written out explicitly. This makes checking easier, and also allows one to build in the correct orientation of the triangles. That is, a normal vector computed from each triangle by the right-hand rule always points in the direction of increasing scalar value.

The interpolation implemented in the marching cubes algorithm is linear on the

edges of the hexahedron. This makes it exact for cubic cells. It falls somewhere between linear and trilinear interpolation for hexahedra, consisting of linear interpolation on the edges. For tetrahedra, multilinear interpolation reduces to linear, since there are only 4 vertices.

The original marching cubes algorithm us two-valued, binary logic at each vertex. Thus the tables have 16 entries for tetrahedra, 256 for hexahedra. A variant for which the value of "zero within tolerance" has also been tried for tetrahedrons (having 3**4 or 81 patterns), and yields better results in some cases, at the cost of more complication. In this work, the two-valued system is used, since it has a much more convenient hardware implementation. For a hexahedron, there are 3**8 or 6561 patterns, so this approach was not tried. A more fruitful direction would be to use tolerance checking to trigger an adaptive subdivision of the surface for cubes which lie very to the isosurface.

The subroutines which perform the table lookup take as input a single cell, and output triangles. In order to conserve memory, it was decided not to keep lists of triangles, but to render them directly as soon as they are produced. If more memory or triangular mesh primitives are available, one should consider putting the triangles into a list and rendering them all at once.

3.3 Sorting and Sweeping

The method described thus far looks at each cell individually and produces a number of polygons depending upon the relative value of the field at the cell vertices compared with the isovalue. The cells are intersected with the desired cutting plane. For each 3-D element in the set, a planar polygon is produced whose vertices carry function values produced by interpolation of the data along the edges of the cell. Transfer functions can be invoked to convert the data value(s) into color, opacity, texture, etc.

For a cutting plane, we expect that for an NxNxN grid, only about NxN cells will actually fall on the cutting plane and be in an "active set" for rendering. Thus a significant savings in rendering the cells can be achieved if we do not need to examine all cells each time we move the isolevel. We are naturally led to the concept of an active set of cells, which can be updated during a sweep through the values of the isoscalar. This is easiest to see in the case of a cutting plane. Given a direction of sweep, once a cell is below the plane, it need never be looked at again.

In order to sweep through the database efficiently, we need to maintain an active set of points. In order to construct the active set it is necessary to preorder the points initially. Assume that the sweep of the data is conducted in order of increasing value of the level surface. A cell will be added to the active set from the MIN list as the isovalue passes its value. A cell will be deleted from the active set once its maximum value has been exceeded. Thus, we exclude all values from the active set which are either completely above or completely below the level surface.

The active set must support the operations of addition of cells and subtraction of cells. The subtractions can take place from any cell in the list. Therefore, it is necessary either to use a linked-list data structure or to keep track of each cell's current position in the active set list. We have chosen the linked-list structure in our implementation.

The foregoing suggests that the cell data structure should contain at least the following information. For the moment we assume that the rest of the data values can be obtained from the existing grid structure by means of the indices I,J,K. In the case of unstructured grids, one keep some other pointer into the list of data values, probably a single index.

```
struct Cell{
float Minval /* Minimum scalar function value at vertex */
float Maxval /* Maximum scalar function value at vertex */
int I,J,K    /*  indices into curvilinear grid          */
Cell*  next_Cell}
```

These cells are presorted into two simple arrays called MIN and MAX. If we begin below the minimum level value, the active set is empty. Imagine that we are going to sweep in the direction of increasing isovalue. Then we incrementally add cells and delete cells from the active set in a loop.

```
procedure sweep_up (bottom_level, top_level, delta)
     level = bottom
     initialize_active_set
     while (level < top_level)
           increment_level (level, delta)
     endwhile
end sweep_up
```

The active set is maintained by first adding cells to the active set (at the tail of the linked list structure) from the MIN list, which has already been preordered. The we filter out those cells whose maximum value lie below the threshhold and which never need to be examined again. Thus the active set is maintained by

```
procedure increment_level(level, delta)
     level = level + delta
     add_cells_from_MIN_list
     delete_active_cells_by_max(level)
     render_active_set
end  increment_level
```

and

```
procedure delete_active_cells_by_max(level)
     for(all cells in active set){
           if(cell->max < level) delete cell
     }
end delete_active_cells_by_max
```

The process to be followed is illustrated in Figures 2 and 3. The cells have been ordered by their min and max values. As the level is increased, the threshold encounters new cells from the MIN list. These are added onto the tail of the active set, with pointers appropriately updated. Then, the cell maxima in the active set are checked against the current value of the threshhold. Those cells whose maxima are below the threshhold are then deleted from the active set by resetting the links and freeing the memory allotted. The

deletion operation can be performed in one pass over the active set.

Naturally, one can reverse the process and sweep down from the top. In that case, the roles of the MIN and MAX lists and values are reversed. Only the MIN list is required if the dataset is swept in only one direction.

Since the list of cells to be sorted is very large, we need an efficient method of sorting the cells and maintaining an active list of cells. The present method uses a variant of the HEAPSORT algorithm [PRES88 based on KNUT73] which is O(N log N). The QUICKSORT algorithm may be faster on large datasets but has not been implemented. O(N**2) sorts are out of the question for these large datasets!

The sorting and sweeping described here bears a (deliberate) analogy with the methods currently in use to do scan-line rendering of polygons in hardware. There the problem is to keep an active set of polygons which is rendered if it crosses a scanline. In either case, one keeps trackof a minimum and maximum value for the polygons before which and after which, the polygon is not intersected by the scanline. More details about such algorithms, and clues to their implementation in hardware, can be found in [FOLE82]. For a more theoretical treatment of sorts and sweeps in computational geometry, see [PREP85] and [EDEL87].

4. Additional Visualizations

Three color plates of these visualizations are included at the end of this paper. Besides the visualizations of isosurfaces and cutting planes, some additional visualizations are possible using these techniques. This is not an exhaustive list, as most successful visualizations involve a combination of techniques. Since we are creating vectors and polygons, one can combine these elements, together with lighting, shading, stereo, and motion to produce a final product.

4.1 Multiple Isosurfaces.

Plate 1 illustrates a typical screen view of the ISOLEV program showing an iso-pressure surface (isobar) for the blunt fin dataset generated by Hung and Buning [HUNG84]. Since the isosurfaces are produced as geometric objects, one can compose them either with isosurfaces corresponding to different values of the constant β, or with isosurfaces of a different function. Generally, one seeks to relate the behavior of two variables through the representation by their isosurfaces.

4.2 Multiple Cutting Planes

Plate 2 shows a level surface of the same pressure data for a blunt fin. The cutting plane is at an arbitrary angle, and a color transfer function represents the pressure. If we define a level surface by a planar function, we can use both color and transparency to encode scalar variables. The use of transparency makes it possile to show multiple cutting planes in the same image. This has been done in some examples with DPLANE to good advantage. Typically, one uses a transparency function to eliminate "uninteresting" parts of the flowfield (like the freestream) from view. Unfortunately, the present generation of firmware does not permit smooth interpolation of opacity (analog of Gouraud shading) for opacity.

4.3 Vector Fields

Besides scalar fields, there are also vector and tensor fields which occur in nature and which are simulated or measured, and the visualization of these fields can be aided by variations on the techniques described above. In particular, one realizes that any dependent variable or set of variables defined at the vertices of the discretization cells can be linearly interpolated to the vertices of the polygonal isosurfaces or level surfaces. Thus vector- or tensor-valued functions have a unique interpolant obtained by interpolating component by component.

4.3.1 Vector plots.

The simplest plot is the traditional 3-D vector plot with the magnitude of the graphical vector being proportional to the magnitude of the vector field. Thus one can produce sets of 3-D vectors which emanate from a chosen cutting plane or surface.

Alternatively, one can visualize only the vector directions in a manner analogous to the tuft grids used in experimental flow visualization. However, the array of initial points is not a regular grid but consists of the irregular array of triangles which forms the cutting plane. One could obtain values at a regular grid by a second set of interpolations, of course.

4.3.2 Deformation surfaces.

A related method of visualizing vector fields is to produce a vector emanating from every vertex on the cutting plane, and then create polygons from the endpoints of the vectors. If the vector field under study represents a displacement field, then the surface so produced is a deformation surface due to the displacement. One can scale this displacement continuously from zero in order to add motion cues. A deformation surface corresponding to velocity is shown in Plate 3. One also has the transfer functions of color and opacity at one's disposal in handling these vectors. A common practice in CFD visualizations is to color velocity vectors by pressure , as is done here, or by Mach number.

4.3.3 Tensor fields on level surfaces.

A tensor field can usefully be described as mapping from vectors to vectors. Thus, given one set of vectors, the normals to a given cutting plane, one obtaines a second set of vectors by contracting the normals with a tensor field. One can then sdisplay the resulting vector field using the methods just described. By using multiple sets of planes, or using one plane interactively, one gain some impression of a few components of a tensor field at a time.

5. Future directions

The methods described above use only the simplest forms of interpolants to discrete data. These forms will probably predominate in the near future, if only because most simulations only solve the underlying differential equations to this order of accuracy. However, we can describe some probable extensions to these methods as follows.

5.1 Higher Order Interpolants

The natural generalization of the concepts derived here is to higher order interpolants which can exhibit higher degrees of parametric or geometric continuity.

Continuity conditions have been worked out for one-dimensional space curves and to a lesser extent for two-dimensional surfaces. The study of 3-D volume interpolants is still at an early stage. One would expect the isosurfaces of higher order interpolants to inherent the continuity properties of its basis, but this has not been shown. As far as methods for determining these interpolants, one must generally solve for roots of algebraic equations of the order of the interpolant (i.e. solve simultaneous cubic equations for intersection points of level surfaces with cubic interpolants) or use some sort of iterative or adaptive sheme. It might be possible to use a considerably more elaborate table lookup scheme which involves a much larger list of possibilities. Sorting is no more difficult in principle, because higher order spline functions are contained within the convex hull of their control points in the Bezier representation.

5.2 Adaptive Subdivision

We have already mentioned before that, depending upon the proximity of the vertices to the isosurface, some inaccuracies in the surface may result. These can usually be improved by subdividing the cells. At present, only an overall subdivision (i.e. for every cell) is implemented, but one could include a test for each cell which would determine under what conditions the cell should be subdivided. In principle, this could included information about the projected image as well as the interpolants.

5.3 Vectorization and Parallelization

The sorting routines introduced earlier presuppose serial processing of cells. It would be possible to use hardware parallelization on some of these processes, but this has not been investigated here.

6. Conclusions

A computer program has been written which adds to the repertoire of visualizations that can be created with the aid of lookup tables. Visualizations include isoscalar surfaces, cutting planes, and vector deformation surfaces. Clearly, the techniques is a fruitful source of ideas for visualizing continous fields generated as interpolants of spatial data.

7. Acknowledgements

Some of these ideas have been used in other packages. The surface-on-surface routines were available in the ARCGRAPH mesh library (albeit with some of the bitmaps missing).

The original set of bitmaps was obtained from Marty Fouts, then of the NAS Systems Development branch, and subsequently modified by the author. for general topological cubes with revised cube nomenclature and bitmaps. Forms of the bitmaps for tetrahedra were tested out in earlier programs by the author for visualization based on Delaunay triangulations.

After these ideas were developed, the author became aware of similar ideas in [GILE89] which were developed independently.

8. References

ARCG88 Makatura, G., and Hibbard, E., ARCGraph Mesh Library documentation, NASA Ames , 1988.

DAVI85 Davies, D.E. , and Salmond, Deborah J., *AIAA Journal,* Vol 23, No. 63 (1985) pp 954-956.

DEFL87 DeFloriani, L. "Surface representations based on triangular grids," *The Visual Computer,* Vol 3 (1987) pp 27-50.

DREB88 Drebin, R.A., Carpenter, L., and Hanrahan, P., "Volume Rendering," *Computer Graphics* Vol 22, No. 4, August 1988, pp. 65-74.

DUER88 Duerst, M.J. "Letter: Additional Reference to 'Marching Cubes'" *Computer Graphics* Vol. 22 No. 2, April 1988. Indicates an example where the MC method leads to a hole in an isosurface.

EDEL87 Edelsbrunner, H., "Algorithms in Computational Geometry," Springer-Verlag, Berlin 1987.

FOLE82 Foley, J.D., and van Dam, A., "Fundamentals of Interactive Computer Graphics," Addison-Wesley, Reading, Mass, 1982.

FREN89 Frenkel, K.A., "Volume Rendering" *Comm. ACM,* Vol. 32 No. 4, April 1989, pp 426-435.

GILE89 Giles, M. "Algorithm for 2D Viewing Plane Construction" preprint dated March 22, 1989, This one-page note describes in cursory fashion some of the issues raised here, in particular the use of "min" and "max" keys on the cells to facilitate

the keeping of the "active set" of volumes to be checked.

HUNG84 Hung, C.-M., and Buning, P., "Simulation of Blunt-Fin Induced Shock Wave Boundary Layer Interaction," AIAA Paper 84-0457, AIAA 22nd Aerospace Sciences Meeting, Reno, NV Jan. 9-12, 1984.

KNUT73 Knuth, D.E. "Sorting and Searching," vol. 3 of "The Art of Computer Programming," Addison-Wesley, Reading, Mass, 1973, pp. 146ff.

LEVI88 Levit, Creon, C_Plane program, personal communication.

LEVO88 Levoy,M.S., "Volume Rendering: Display of Surfaces from Volume Data, *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May 1988.

LORE87 Lorenson, W.E., and Cline, H.E., "Marching Cubes: a High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol 21., No.4 July 1987, pp. 163-169.

PLOT89 PLOT3D User's Manual, Workstation Applications Office, NASA Ames, March 1989.

PREP85 Preparata, F.P., and Shamos, M.I., "Computational Geometry An Introduction" Springer-Verlag, Berlin 1985.

PRES88 Press,W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., "Numerical Recipes in C,", Cambridge University Press, 1988.

TRIS88 Tristram, David, D_Plane program, personal communication.

TRIS89 Tristram, D.A., and Walatka, P.P, "Panel Library Programmer's Manual", NASA Ames Research Center, NAS Division preprint, April 1989.

UPSO88 Upson, C., Keeler, M., 'V-BUFFER: Visible Volume Rendering', *Computer Graphics*, Vol. 22, No, 4, August 1988, pp. 59-64.

UPSO89 Upson, C., and Kerlick ,G.D. "Volumetric Visualization Techniques," *1989 ACM SIGGRAPH Course #13*, Boston, MA 31 July 1989.

VISC87 McCormick, B.H., DeFanti, T.A., and Brown, M.D., "VIsualization in Scientific Computing" *Computer Graphics* Vol. 21 Number 6, November 1987.

WEST89 Westover, L., "Interactive Volme Rendering" in Proceedings of "Chapel Hill Workshop on Volume Visualization," C. Upson, ed., Chapel Hill, NC., May 18-19, 1989.

WYVI86a Wyvill,G., McPheeters, C., and Wyvill, B., "Soft Objects." Proceedings of Computer Graphics Tokyo '86 (Tokyo April 22-27, 1986). In "Advanced Computer Graphics," T.L. Kunii, ed. Springer-Verlasg, Tokyo, 1986, 114-128

WYVI86b Wyvill,G., McPheeters, C., and Wyvill, B., "Data Structures for Soft Objects." *The Visual Computer*, Vol 2 No. 4 (Aug 1986) 227-234.

9. Figure Captions

Figure 1. Computational Hexahedron or topological cube. This figure is defined by 8 vertices connected by 12 edges. The faces are not necessarily planar, but are doubly ruled surfaces of zero Gaussian Curvature (see for example [DAVI85]).

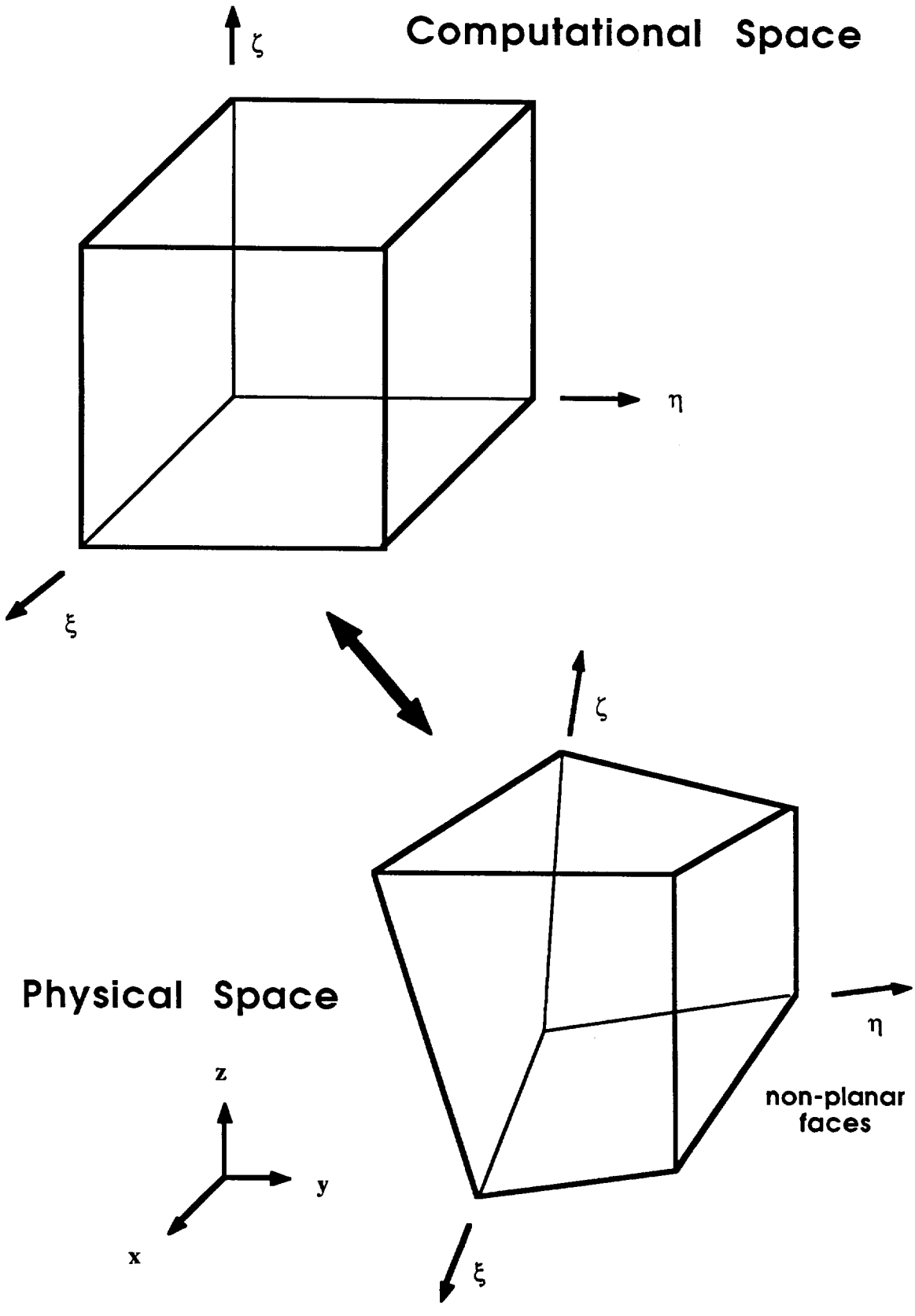
Figure 2. Cells are projected onto a line in the normal direction. They can thus be represented for the purposes of sorting by their minima and maximum projection on the normal line.

Figure 3. Segment lists. In this figure, the process of maintaing the active set of cells is summarized. The threshhold is represented by the horizontal level line which is rising through the set of cells. As the level rises, both the MIN and MAX lists are incremented . The active set is the difference between the MIN and MAX lists. Active elements are here indicated by thicckened lines.

Plate 1. Pressure Isosurface for the blunt fin dataset. Two windows show the isosurface in both physical space and computational space.

Plate 2. Level Surface of the blunt fin data set colored by pressure.

Plate 3. A level surface in the streamwise direction has been colored by pressure and deformed by the velocity field. At the fin and plate, the velocity, and therefore the deformation, vanishes.

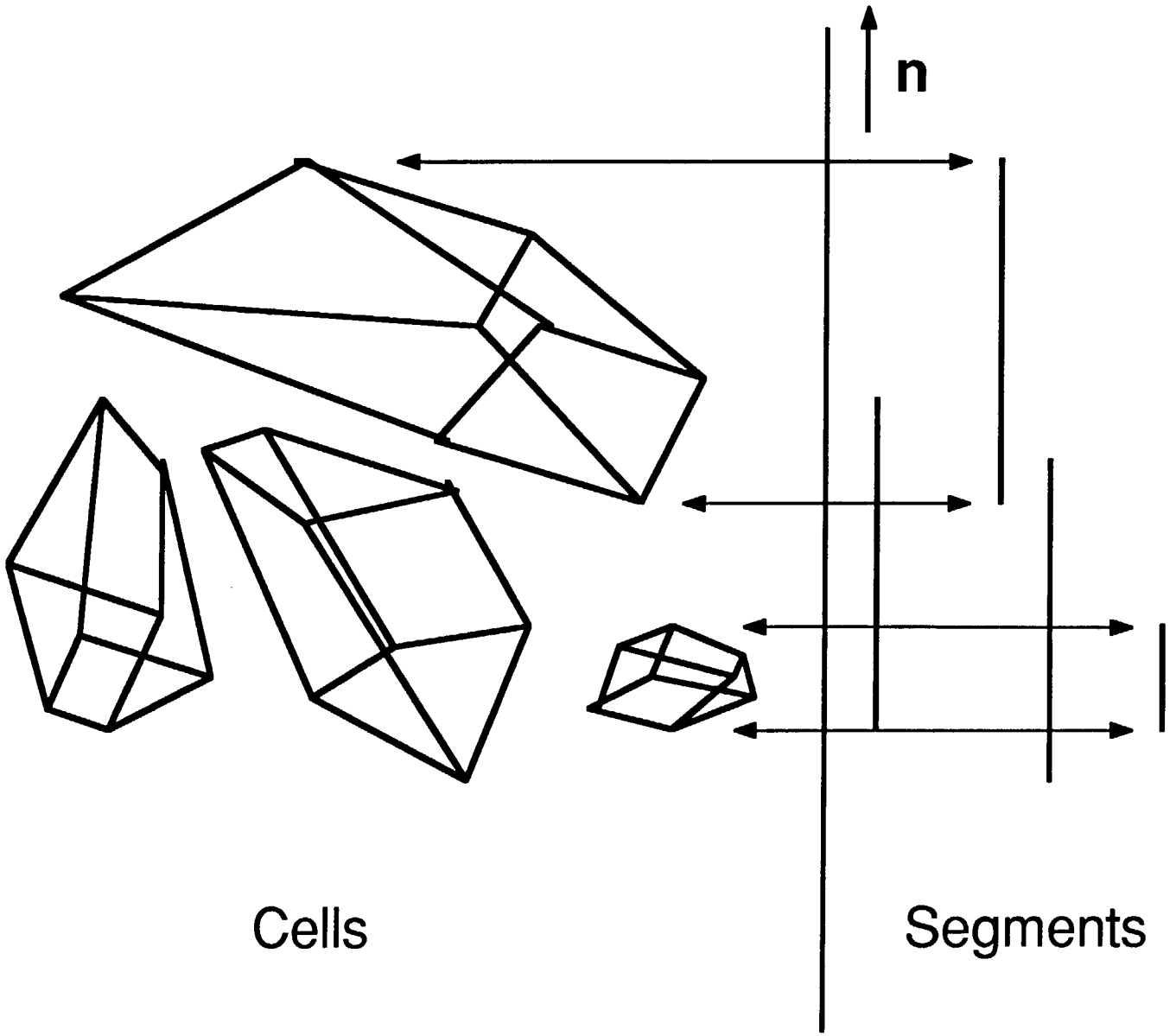


Computational Cube / Hexahedron

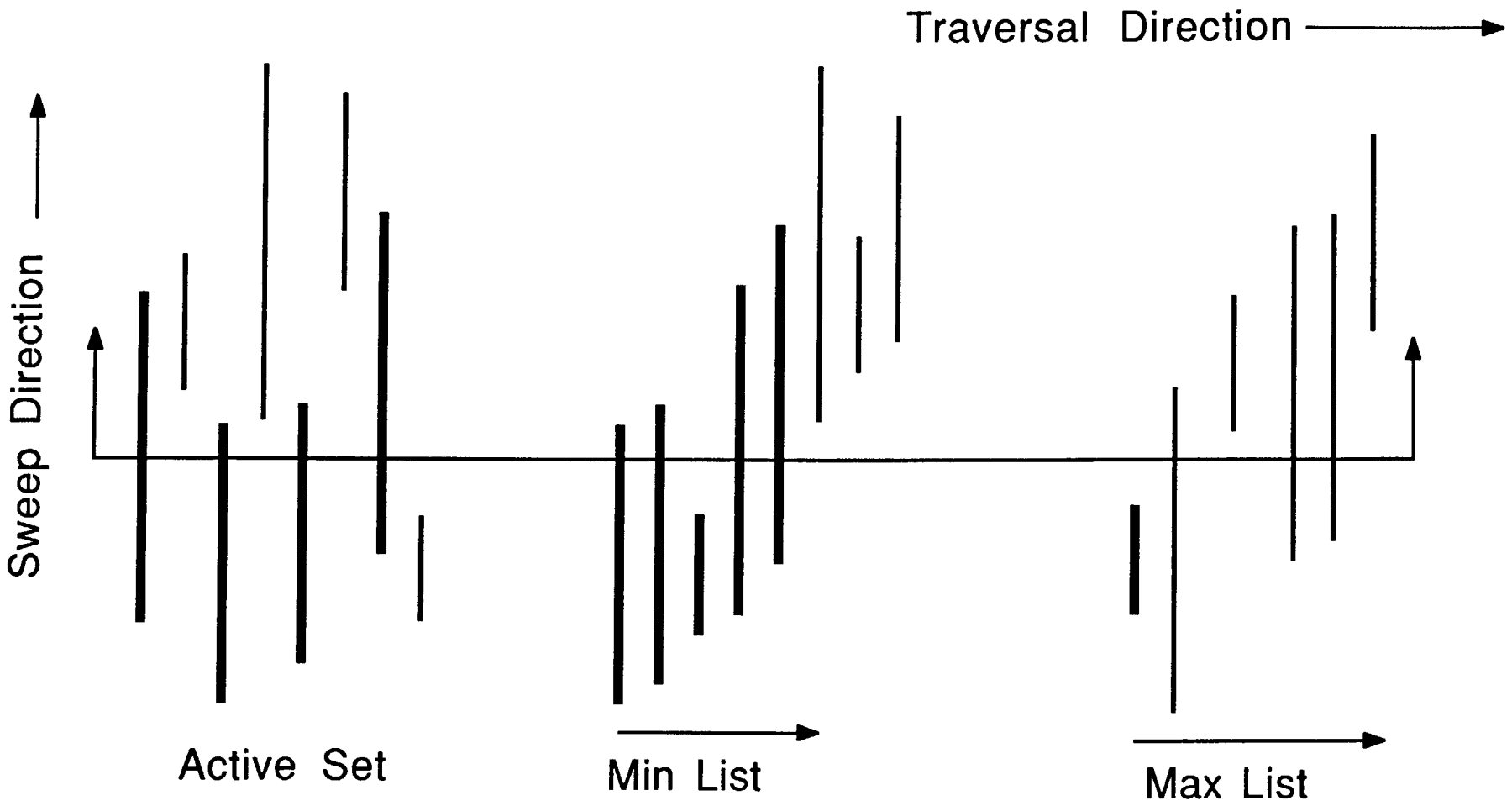
RNR-089-006

Kerlick 2/89

Cells Projected Into Segments



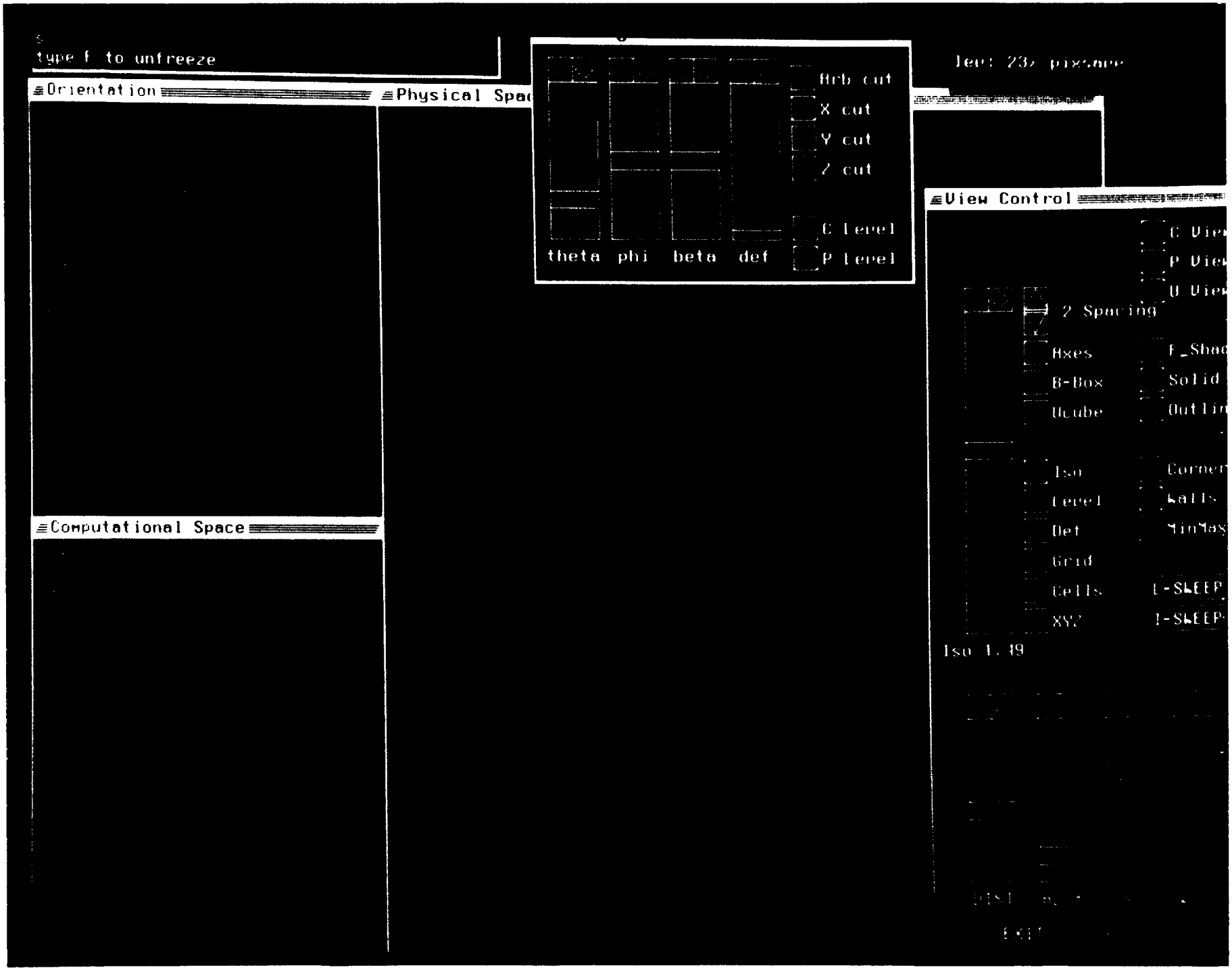
Active Set of Segments



Active Set = (Cells Added from Min List) - (Cells Delete from Max List)

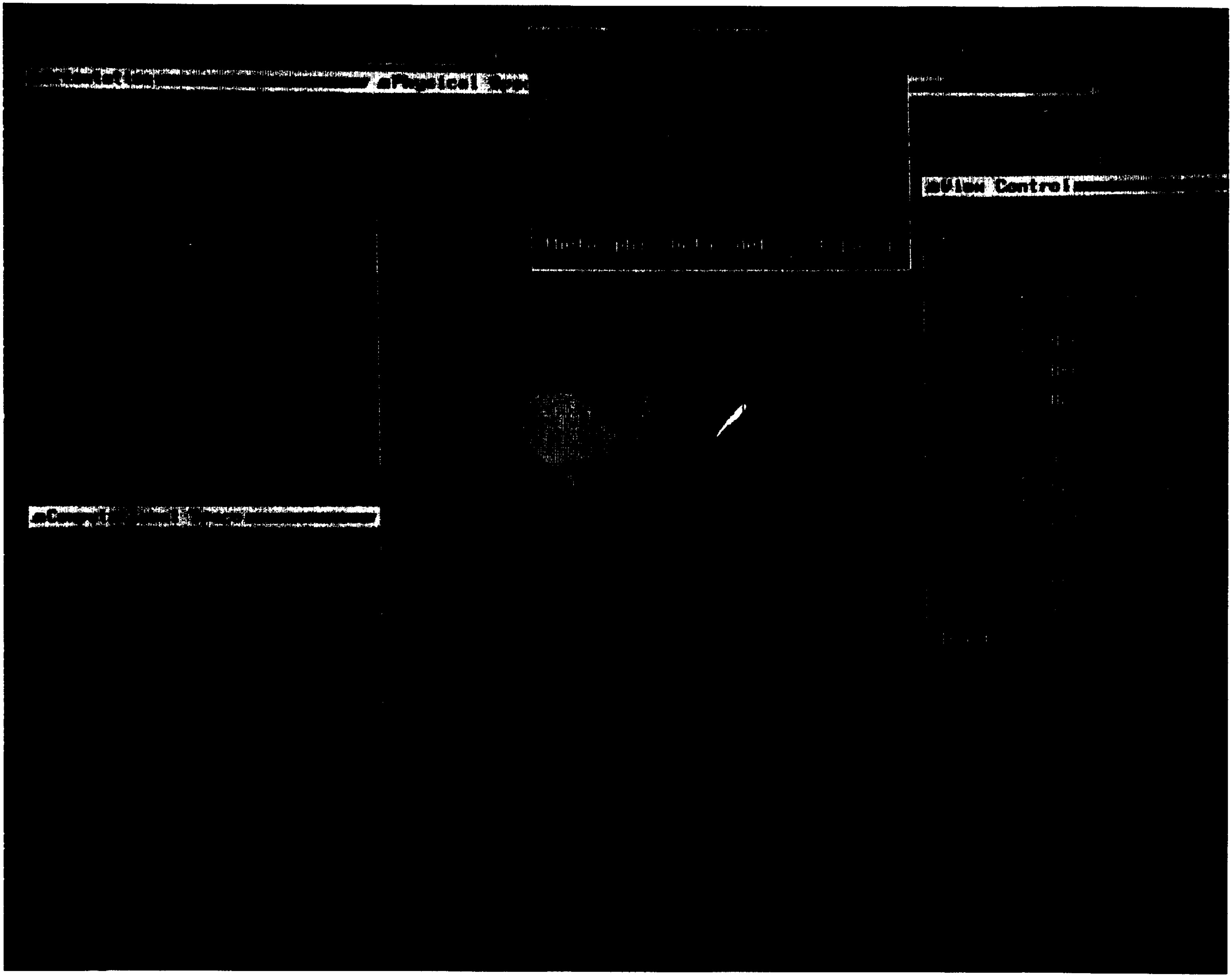
RNR-089-006 Fig 2

Kerlick 4/89



RNR-89-006 plate 1 ISOLEV D Kerlick

Isobars of pressure blunt fin



RNR-89-006 Dinho 7° Level Surface ISOLEV D Kerlick

RNR-89-006 Plate 3 ISOLEV D. Kerlick Velocity field deforms surface, colored by pressure blunt fin